Dungeon Crawler Mobile

Pol Genis Romero Virue

Resum—Dungeon Crawler Mobile és un joc 2D per a dispositius Android de gènere Dungeon Crawler, on el jugador haurà d'avançar a través de sales plenes d'enemics, generades de forma procedural, fins a que aquests (els enemics) el derrotin. Per a vèncer els enemics, el jugador té a la seva disposició una sèrie d'atacs i habilitats, així com també pot recollir objectes que milloraran aquestes habilitats. Els controls del joc han sigut dissenyats tenint en compte les limitacions dels dispositius mòbils, així com també la dels jugadors. El projecte ha sigut realitzat fent servir Kotlin com a llenguatge de programació i LibGDX com a framework per a la visualització. La arquitectura emprada pel projecte ha sigut MVP amb els principis del Clean Code per tal de permetre una modificació i manteniment més senzills.

Paraules clau—videojoc, 2D, Android, Kotlin, MVP, model-view-presenter, Clean Code, LibGDX

Abstract—Dungeon Crawler Mobile is a 2D game of the Dungeon Crawler genre for Android devices. The Player has to advance through a number of enemy filled rooms, generated procedurally, until they (the enemies) defeat him (Player). To defeat the enemies the Player has at his disposal a number of attacks and abilities, as he also can recollect objects that will boost those abilities. The controls of the game have been designed Accounting for the limitation that Mobile devices present as much as those present when the user plays. The project has been developed using Kotlin as its programming language and LibGDX as the framework to render the visuals. The architecture used for the project has been MVP with the Clean Code principles as to allow an easier time when modifying or maintaining the game.

Index Terms—videogame, 2D, Android, Kotlin, MVP, model-view-presenter, Clean Architecture, LibGDX

1 Introducció

AQUEST projecte consisteix en crear un joc del gènere *Dungeon Crawler*[1] per a dispositius de sistema operatiu Android. L'objectiu d'aquest joc és superar el màxim d'habitacions possibles, on el jugador s'enfrontarà a enemics que l'intentaran vèncer i ell els haurà de derrotar per tal d'accedir a la següent habitació.

El article es divideix en sis seccions. Primer una introducció on s'expliquen els orígens, la motivació i els objectius del projecte. Segon una secció on s'explica la metodologia emprada per al desenvolupament. Posteriorment una explicació del desenvolupament. Seguit dels resultats del projecte i finalment les conclusions.

1.1 Origen i Motivació

Els primers jocs de gènere *Dungeon Crawler* apareixen durant la dècada dels 80 després del esclat de popularitat dels jocs de rol de taula, en especial *Dungeons & Dragons*. En aquest jocs de rol, de paper i llapis, existeixen uns escenaris anomenats *dungeon crawl*[1] on els jugadors exploren un castell, cova o masmorra en busca de completar un objectiu o amb l'única intenció d'explorar en busca de recompenses. Aquest tipus d'escenari intenten crear una sensació d'exploració i descobriment en els jugadors.

Poc després van aparèixer els primers jocs del gènere, tals com *Wizardry*, *The Bard's Tale* o *Might & Magic*[2], que van traduir aquests escenaris i sensació d'exploració i descobriment al medi dels videojocs. Aquests jocs van assentir la base del gènere amb una estètica on la repetició del entorn era utilitzat per a crear un espai laberíntic. Al mateix temps que aquests títols consolidaven el *Dungeon Crawler* també va aparèixer un subgènere d'aquest, anomenat *Roguelike*, caracteritzat per la generació procedural de nivells i la pèrdua de tot progrés que el jugador hagués aconseguit en el joc en cas de la mort del seu personatge.

En l'actualitat aquest gènere de jocs segueix sent de nínxol i principalment desenvolupat pel mercat indie (petit grups de desenvolupadors). Encara que segueixen havent entregues de clàssics com els anterior mencionats *The Bard's Tale* i alguns jocs arriben al gran públic com pot ser la sèrie *Persona*[3]. Mentrestant els *Roguelike*, encara i no arribar al gran públic i ser únicament desenvolupats per estudis indie, s'han popularitzat en un mercat on tota una generació de jugadors busca reptes en la seva dificultat en un mercat on la norma és facilitar-ho tot per tal d'arribar a un major nombre de jugadors

Aquesta mentalitat de crear jocs senzills i fàcils és especialment estesa en el mercat dels dispositius mòbils. És aquí, on els desenvolupadors i distribuïdores els utilitzen com a reclam per atreure a persones que la seva única experiència en el medi es a través d'aquest mercat per a posteriorment enganxar-los en la seva agressiva

- *E-mail de contacte: polgenis.romero@e-campus.uab.cat*
- *Menció realitzada: Enginyeria del Software.*
- *Treball tutoritzat per: Javier Sánchez Pujadas*
- *Curs 2018/19*

monetització. És degut a aquesta falta de títols més complexes sense una monetització agressiva el que motiva aquest projecte a ser per dispositius mòbils.

1.2 Objectius

L'objectiu principal del projecte és desenvolupar un joc del tipus *Dungeon Crawler* per Android. Amb aquesta premissa es van establir els següents objectius:

- Creació de una interfície de joc capaç de ser utilitzada per l'usuari amb una sola mà i que transmeti la informació necessària al jugador de forma senzilla.
- Addició d'una sèrie d'enemics que creïn un repte per al jugador i objectes que l'ajudin a derrotar dits enemics.
- Implementació de una base de dades que mantingui un estat entre partides del jugador.
- Disseny i implementació d'un sistema de combat satisfactori per al jugador.
- Dissenyar i implementar l'aplicació amb una arquitectura que segueixi els patrons i disseny de *model-view-presenter*[4] (MVP) i *Clean Architecture*[5].

1.3 Estat de l'art

Des de la creació dels primers *Dungeon Crawler* aquest ha sigut un gènere dirigit primordialment al públic que juga en un ordinador. Actualment els jocs amb més èxit són *Diablo*[6] i *The Binding of Isaac*[7]. Aquests dos, a pesar de haver sigut llençats per a diverses consoles, van ser dissenyats per a una experiència d'ordinador la qual dificulta la jugabilitat en les altres plataformes. Cadascun d'aquests jocs té un sistema de combat diferent però mantenen aspectes bàsics iguals com la recol·lecció d'objectes.

Mentre al àmbit de dispositius mòbils han aparegut jocs com *ReDungeon*[8] o *Pixel Dungeon*[9], tots dos en 2D. Aquests jocs van ser dissenyats tenint en compte les limitacions del dispositius mòbils així com la dels jugadors al utilitzar-los. Tots dos empren un sistema de control basat en tocar en la pantalla per a les accions més senzilles o desllizar el dit, aquesta ultima especialment per a aquelles accions que en cas de ser implementades amb la primera serien massa llargues. Aquest esquema de controls està dissenyat per a que el jugador pugui jugar amb una sola mà. Permetent així una bona experiència de joc sense importar on l'usuari es trobi. Tots dos, igualment pel mateix motiu, comparteixen una vista superior per a poder transmetre al jugador el màxim d'informació possible.

2 Metodologia

Per a realitzar el projecte s'ha emprat una metodologia agile iterativa. El treball s'ha dividit en iteracions que agrupaven i atacaven una faceta específica del desenvolupament del joc.

Les iteracions han seguit un procés específic. Primer s'ha generat la documentació necessària per al desenvolupament de la iteració, com pot ser el diagrama de classes. Posteriorment s'ha dut a terme la implementació del codi necessari per a complir la iteració. Finalment s'han dut a terme una sèrie de tests marcats per comprovar que la implementació ha sigut correcte, i en cas de no ser així s'ha revisat el codi fins que el test s'ha passat.

3 Desenvolupament

El desenvolupament del projecte es va dividir en els apartats explicats a continuació.

3.1 Arquitectura

Per a desenvolupar el projecte es va decidir per utilitzar *model-view-presenter*, una arquitectura derivada del *model-view-controller* que s'ha convertit en un estàndard en el mercat de les aplicacions mòbils degut a la facilitat per a la addició i al manteniment de les aplicacions. De igual manera en l'arquitectura s'utilitzen els principis del *Clean Code*, una sèrie de regles que al igual que el *model-view-presenter* han sigut adoptades com a estàndard en un mercat on el desenvolupament més ràpid possible i la facilitat de manteniment són realment necessaris.

En la arquitectura de tipus *model-view-presenter* n'hi han tres parts:

- *Model:* classes que guarden informació i la lògica per la interacció entre elles.
- *View:* mostra al usuari la interfície i reacciona a els inputs del usuari. No posseeix cap tipus de lògica que no estigui lligada amb la visualització per part del usuari. Només pot interactuar amb el *presenter* a través de una interfície. En el cas del projecte la *view* són aquelles classes que controlen la visualització de sprites per pantalla.
- *Presenter:* controla la lògica del programa utilitzant el *model* i reacciona als inputs de la *view*.

Aquesta divisió permet que en cas de necessitar canviar el *view* o el *controller* es pugui fer amb certa facilitat. Alhora degut als principis de *Clean Code* es divideix en dos parts:

- *Domain*: lògica de negoci de l'aplicació, així com classes necessàries per aquesta. En aquesta part del *model* és on es troba tota la lògica del joc i totes les classes necessàries per a ella.
- *Data*: part del codi que interactua amb qualsevol tipus de font de dades, tal com bases de dades o trucades a serveis de REST APIs.

Cap d'aquestes dues parts té coneixement de l'altre exceptuant per interfícies. Aquesta addició al *model-view-presenter* permet que, sempre que no sigui el *domain* el que s'hagi de reemplaçar, es pugui canviar sense gaires problemes. Permetent així en llenguatges de programació natius multiplataforma com l'utilitzat en el projecte, Kotlin[10], aprofitar part del codi en diversos sistemes operatius. (Diagrames de classe separats al Apèndix A2)

Com s'ha mencionat prèviament, aquesta arquitectura aporta un gran valor a la mantenibilitat i facilitat d'expansió del codi. És per això que s'ha seleccionat dita arquitectura, també ha sigut un factor que el joc no sigui gens exhaustiu per als processadors actuals dels dispositius mòbils. Si aquest no fos el cas, i s'hagués necessitat una major quantitat de rendiment, s'hauria escollit una arquitectura més senzilla.

El primer pas que es va fer per tal d'implementar

l'arquitectura va ser adaptar el disseny per al projecte i generar un diagrama de classes.

L'arquitectura com es pot observar en la *Fig. 1* és altament modular permetent així que es facin adicions de forma senzilla una vegada el primer objecte de cada classe ha sigut implementat.

Posteriorment a la generació del diagrama de classes es va implementar la arquitectura. A l'hora de crear el codi únicament es van implementar les classes abstractes i interfícies de les que posteriorment la resta de classes heretarien.

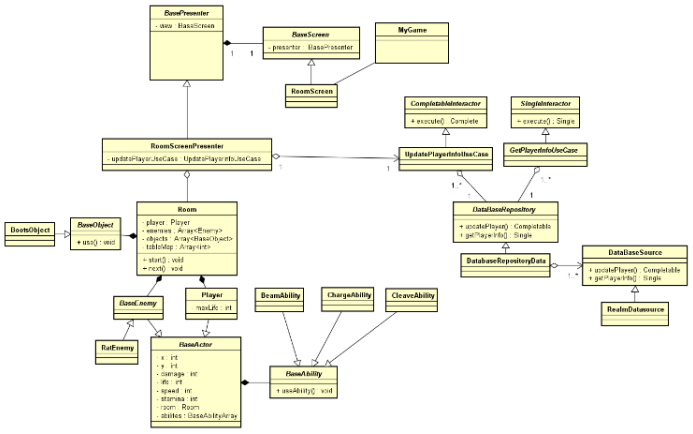


Fig. 1 Diagrama de classes del projecte.

3.2 Visualització

Una vegada es va finalitzar l'arquitectura el següent pas va ser crear la *view*, es a dir la part que genera la part visual del joc. Per aquesta tasca es va emprar la llibreria LibGDX[11]. Aquesta llibreria fa d'interfície amb OpenGL[12] per tal de mostrar els *sprites*, o també anomenats textures, que es desitgen.

Fig. 2 Exemple de múltiples casella

La visualització del tauler es genera mitjançant la utilització de la casella mostrada en la Fig. 2. A sobre d'aquest tauler llavors es mostren els sprites del jugador, els enemics i els objectes seguint un ordre depenent de l'*alçada* en la que estiguin per tal de mostrar-los amb les superposicions correctes.

Aquest sistema funciona tant per a sprites quiets com per animacions, ja que la única diferencia entre un i l'altre és el sprite que es desitja mostrar cada milisegon. Per fer les animacions únicament s'ha hagut que calcular quan canvia de *sprite*.

Fig. 3 Exemple de sprites d'animació

3.3 Joc

El desenvolupament de la lògica del joc es va fer per fases. Primer es va desenvolupar el jugador i els torns, posteriorment els enemics i finalment els objectes.

3.3.1 Jugador

El jugador té com a objectiu vèncer enemics per tal d'avançar a una altre habitació. Amb aquest fi pot fer diverses accions:

- Atacar
- Utilitzar Habilitat
- Recollir Objecte
- Moure

Part d'aquestes capacitats del jugador es van delegar a la classe *pare* del jugador *BaseActor*. Això ha permès que es reaprofiti tot aquest codi que també utilitzaran els enemics.

3.3.2 Torns

El joc es basa en un sistema de torns amb temps límit on el jugador i els enemics fan el seu moviment a la vegada. És una simplificació del sistema de temps real per a tal de donar una sensació d'estratègia al joc.

Aquest sistema de torns concurrent s'ha implementat mitjançant un thread apart on s'espera la finalització del temps del torn o el input del usuari per tal de executar el càlcul de les accions dels enemics.

3.3.3 Enemics

Els enemics són l'obstacle que el jugador ha de superar per tal de millorar la seva puntuació. Aquests enemics tenen diverses habilitats i patrons diferents.

Part de la implementació dels enemics ve donada per la seva classe *pare BaseEnemy* i de la que hereta aquesta *BaseActor*.

Per tal de mantenir la il·lusió de que el usuari i els enemics fan els moviments a la vegada, aquests calculen els seus moviments depenent de la posició actual del jugador (ja que fa el moviment primer) i la posició del jugador en el torn previ.

3.3.4 Objectes

Els objectes són millores que el jugador pot recollir per

tal de millorar les seves estadístiques. Millorar les estadístiques del jugador significa que a aquest el serà més fàcil vèncer als enemics en el futur. Les estadístiques son guardades per a partides posteriors en la base de dades, així l'usuari pot avançar a habitacions més difícils en un menor temps.

També hi ha un objecte especial, la *poció de vida*. Aquest objecte a diferencia de la resta no millora les estadístiques del jugador.

3.4 Bases de Dades

El joc té diversos valors que es mantenen per a futures partides. De igual manera també s'han de mantenir les puntuacions del jugador i els patrons d'habitacions que s'utilitzen per a la generació procedural. Per tal de mantenir dits valors s'ha hagut d'implementar una base de dades.

Com a base de dades s'ha utilitzat *Realm*[13], una base de dades no relacional dissenyada per a Android que aporta una gran millora de rendiment comparada amb les altres opcions. *Realm* és una base de dades basada en objectes, el que significa que es guarda tota la informació mantenint la mateixa estructura de dades que s'utilitza en l'aplicació.

En la base de dades es guarden les estadístiques del jugador, així com les puntuacions i les bases per a la generació procedural d'habitacions.

3.5 Generació procedural d'habitacions

La part més important del joc és la generació procedural d'habitacions. Tenir una generació d'habitacions en comptes d'un nombre finit d'habitacions permet que el jugador es trobi en situacions diferents durant un major període de temps. També permet que l'usuari jugui durant tant de temps com ho desitji.

La generació d'habitacions no és completament aleatòria però tampoc es determinista. Per a generar una habitació l'algoritme utilitza una base. Les bases són taulers dissenyats manualment i que tenen una dificultat màxima senyalada. En aquests taulers estan marcades certes caselles on poden aparèixer enemics o objectes. En les caselles d'enemics s'afegirà un enemic aleatori que amb el sumatori del rati de dificultat de tots els anteriors enemics generats, no superi la dificultat marcada de l'habitació. A diferencia dels enemics, els objectes tenen un rati d'aparició depenent de les estadístiques del jugador, a més a més de que per a cada casella d'objecte n'hi ha un 50% de probabilitat de que no es generi cap. (Veure Apèndix A3)

Per a calcular la dificultat màxima a la que un jugador s'enfrontarà, i així escollir una base aleatòria que tingui una dificultat igual o menor, s'ha creat una operació que la calcula tenint en compte les estadístiques del jugador. Aquesta funció calcula la dificultat donant un pes a cada estadística del jugador depenent de lo valuosa que es consideri.

4 Resultats

El resultat del projecte ha sigut un joc completament funcional per a Android, on el jugador té tres vides i ha de superar habitacions per a millorar la seva puntuació.

El joc comença amb un menú on es dona la possibilitat al jugador de començar a jugar o veure les puntuacions.

Fig. 4 Pantalla d'inici

Si el jugador selecciona l'opció de *Score*, llavors se l'envia a una altre pantalla on pot veure les puntuacions.

En la pantalla de puntuacions el usuari pot veure les seves 5 millors puntuacions. Com es pot observar en la *figura 5*, les puntuacions es divideixen en 2 parts: *P* (punts) i *R* (habitacions superades). Està dissenyat d'aquesta manera per a que el jugador tingui una millor noció de la seva progressió. Si el jugador pressiona el botó enrere d'Android se'l retorna al menú principal.

Fig. 5 Pantalla de puntuacions

En cas de que el jugador seleccioni la opció de *Play* se'l enviarà a la pantalla de joc.

Fig. 6 Pantalla de joc

En aquesta pantalla el jugador es troba cara a cara amb els enemics. Aquí es mostren el personatge del jugador, els enemics i els objectes, així com el tauler en la part superior de la pantalla i la interfície d'usuari a la port inferior, on es mostren les vides restants del jugador, les habilitats i la *poció de vida*.

El jugador té diverses maneres d'interactuar amb el joc. La primera és clicant a sobre del tauler. Les caselles interactuables tenen colors diferents. Les caselles verdes són aquelles caselles on si el jugador hi clica pot moure's a aquella posició, mentre que les caselles vermelles només es mostren quan un enemic es troba en una casella on el jugador es podria moure i si el jugador hi clica atacarà al enemic que es trobi en ella.

La segona manera d'interactuar és mitjançant les habilitats. Aquestes són els tres primers requadres que es mostren a la barra inferior. Per utilitzar-les, a diferencia de les caselles, el jugador haurà de polsar sobre una d'elles i fer lliscar el dit en la direcció que la vol llençar. Sempre es té en compte com l'habilitat seleccionada aquella on s'hagi clicat abans de lliscar, així que si el jugador llisca sobre totes elles només n'utilitzarà una.

Fig. 7 Indicador de poció

L'última forma del jugador per interactuar és utilitzant la *poció de vida.* Aquest és un objecte que quan el jugador l'utilitza recupera tots els punts de vida que falten . Només té tres usos. El jugador ha de clicar sobre la imatge del objecte per a obtenir aquest efecte. Per mostrar quantes vegades més es pot utilitzar el botó que es clica canviarà d'imatge amb cada ús fins quedar buit, com es mostra a la *figura 7*.

Una vegada el jugador ha seleccionat una d'aquestes accions o ha passat el límit de temps del torn (1 segon), s'executaran animacions per a mostrar quines són les accions que han fet durant el torn tant el jugador com els enemics. Aquestes animacions duren exactament un segon i quan finalitzen comença el següent torn.

Per a poder fer qualsevol d'aquestes accions el usuari les ha de fer mentre el torn està actiu, això es quan es mostren les caselles de colors diferents. Si el jugador intenta fer alguna d'aquestes accions mentre les animacions de les conseqüències del torn anterior s'estan executant el joc no la registrarà i per tant no tindrà cap efecte.

Els torns del joc sempre duren 1 segon amb un marge de 10 milisegons si el jugador no selecciona cap acció a fer i els torn d'animacions triguen 1 segon amb el mateix marge. El còmput que ha de fer el dispositiu es molt lleuger i gràcies a la implementació pròpia de la mostra de sprites es pot reduir el temps, que utilitzant funcionalitats més avançades, el framework pot fer augmentar.

Si els enemics són capaços de fer que el jugador perdi totes les seves vides llavors es mostrarà la pantalla de *mort* o *Game Over*. En aquesta pantalla és mostra una animació del personatge del jugador morint, així com la puntuació del jugador en la partida finalitzada. L'usuari té dues opcions en aquesta pantalla *Retry* i *Menu.* La opció de Retry permet al jugador tornar a començar una partida sense la necessitat de tornar al menú principal i la opció de *Menu,* tal i com indica el seu nom, retorna el usuari a la pantalla inicial de menú. Com es pot observar en la *figura 8*, en aquesta pantalla no es mostra el numero d'habitacions superades. Aquesta decisió va ser pressa amb la idea de que el jugador no sigui conscient de quantes pantalles ha superat i així crear una sensació de millora superior a la aconseguida, ja que quant més hi juga el jugador més difícil és superar les habitacions.

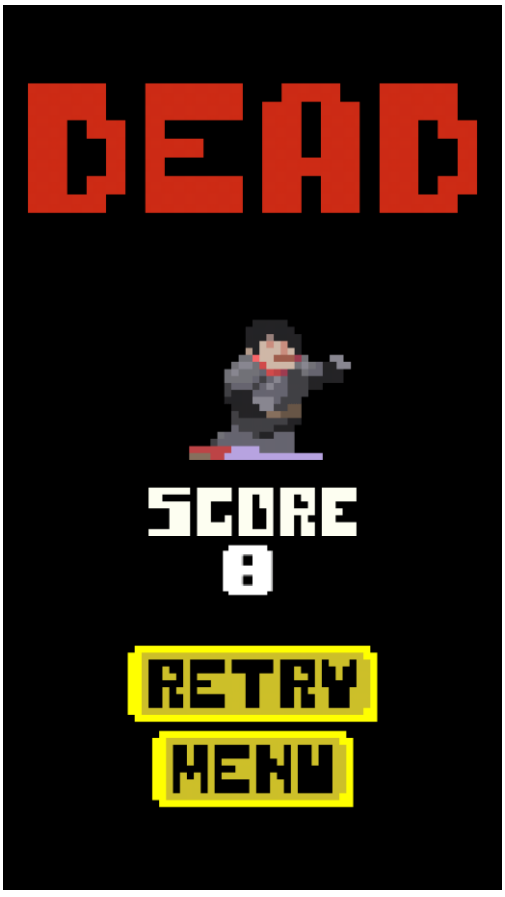


Fig. 8 Pantalla de mort

5 Conclusions

En aquesta secció del informe es presenten les múltiples conclusions arribades durant el desenvolupament del projecte.

5.1 Obstacles i dificultats

El desenvolupament del projecte no ha sigut sense dificultats i obstacles. El projecte inicialment va ser dissenyat com un joc que seria desenvolupat amb el motor gràfic *Unreal Engine*[14]. Això anava a permetre un desenvolupament més ràpid de les parts visuals del projecte. Malauradament el framework per a 2D de dit motor no posseeix una documentació apropiada i per tant es va convertir ràpidament en un gran impediment per avançar. Aquest fet s'ha vist reflexat en el inici del projecte, on es va perdre una gran quantitat de temps amb una eina que no era suficientment flexible i ràpida per al desenvolupament.

La segona major pedra al camí del projecte ha sigut la generació procedural d' habitacions. El problema amb aquesta característica no va ser amb la pròpia implementació d'aquesta si no amb els seus resultats. Durant la part de test posterior a la codificació es va trobar un greu problema de balanç injust amb la generació procedural d'habitacions, on algunes de les habitacions resultants no podien ser completades degut a les habilitats dels enemics que apareixen o la seva quantitat. Els jocs són sistemes complexos on les capacitats del jugador i els seus contrincants han d'esser similar per tal de que la sensació del joc sigui bona i vulgui seguir jugant. En un joc d'aquest tipus és realment important mantenir aquest balanç. Durant la planificació del projecte no es va valorar suficientment aquesta faceta del desenvolupament de la generació d'habitacions i es va haver de modificar la duració de la iteració que la implementava, duplicant el seu temps.

Degut a aquests dos contratemps es van retallar aspectes addicionals que donaven un valor extra al joc. Podem extreure d'aquests daltabaixos que la planificació de un projecte d'aquests tipus necessita un coneixement exhaustiu de les eines que s'emplearan per fer-ho, així com dels riscos específics en aquest tipus de software, que encara i no disminuir de forma tradicional la qualitat d'aquest, són un clar desavantatge en la producció de jocs.

5.2 Avaluació del resultat

Podem dir que el resultat final del projecte ha complert amb els objectius establerts. L'aplicació és un joc del que a pesar de ser simple per a acomodar-se a la plataforma mòbil té una dificultat suficientment interessant pels jugadors que busquen un repte major. És a través de la generació procedural d'habitacions que el joc es capaç d'adaptar-se a les habilitats dels diferents jugadors. Això també permet que el jugador sigui el que controla la durada de la partida, d'aquesta manera s'obre un espai tant a jugadors més dedicats com d'aquells que només desitgen passar una bona estona.

El joc es pot jugar sense dificultat amb una única mà, sempre i quan es jugui en un telèfon de mida normal. Gràcies a aquesta característica es pot jugar en qualsevol moment i lloc sempre i quant l'usuari tingui una mà lliure.

L'arquitectura *model-view-presenter* amb *Clean Code* s'ha implementat i ha representat un repte en el seu ús per al desenvolupament inicial del projecte. Degut a la necessitat de dissenyar i implementar primerament l'arquitectura, per a posteriorment utilitzar-la com a base, ha significat un treball al principi del projecte major que si s'hagués seguit una arquitectura més lleugera i senzilla. Però aquest esforç inicial s'ha traduït en una major facilitat de codificació una vegada s'implementa per primer cop una de les classes. És en casos com els d'aquest joc on l'arquitectura mostra els seus beneficis ja que pot permetre una possible futura versió multiplataforma que aprofiti la part més important del codi.

5.3 Línies futures

Com a continuació del projecte hi ha diverses possibilitats, totes elles facilitades amb l'arquitectura del projecte.

La primera ampliació de joc possible seria la creació de un nou mode de joc. Aquesta millora seria fàcil d'implementar ja que en l'arquitectura del projecte el *presenter* és el que controla com interactuen els jugadors amb el joc i com respon aquest. Per a crear aquest nou mode de joc l'únic que s'hauria de fer és modificar el *presenter* que interactua , en cas de que la *view* es mantingues igual.

La segona evolució del projecte seria millorar la visualització del joc. Donada la poca traça com a dissenyador artístic que té l'alumne, el resultat visual del joc no és el més atractiu de tots. Al igual que amb la primera millora

aquesta només necessitaria de la modificació de la *view* principal del joc. No s'hauria de modificar de cap manera codi que afecti el model de dades utilitzat o cap classe que ho modifiqui.

La línia futura més interessant és l'adaptació del joc a diverses plataformes diferents a Android, o dit d'una altre manera, convertir el joc en una aplicació multiplataforma. Per a poder generar dita versió, el joc hauria d'estar implementat en un framework i llenguatge soportat a les plataformes desitjades, com és el cas de *Kotlin* i *LibGDX* en les plataformes de *iOS* i *Windows*. Donada l'arquitectura sempre en totes les plataformes es podria utilitzar el mateix *presenter* i *domain* (lògica del joc), només s'haurien d'implementar sempre la part de *data* ja que en el cas del projecte s'ha decidit per una base de dades nativa única per a Android. En el cas d'adaptar a una plataforma com és *Windows* que té un format diferent als telèfons mòbils, també s'hauria de modificar la *view* per donar una experiència idònia al jugador

La última ampliació del projecte seria passar tota la lògica del joc a un servidor. Donat que *Kotlin* es pot executar en natiu en la majoria de les plataformes, es pot muntar un servidor que aprofiti la lògica de joc. Seria fins i tot plausible l'ús de la majoria del codi del projecte, només modificant les interaccions que fins al moment la *view* rebia del jugador i transmetia al *presenter*, i que aquest últim les rebés a través del *thread* d'escolta del servidor. Actuant d'aquesta manera com un relay de la *view* del dispositiu on el usuari hi juga. Implementant aquesta adaptació el joc podria obtenir una modalitat cooperativa online.

5.4 Conclusions finals

Com a conclusions finals de la realització del projecte es podrien establir diversos punts que s'han fet aparents durant el desenvolupament.

Durant el projecte s'ha evidenciat que el desenvolupament de un joc és una tasca complicada. Es tracta de un sistema complexa i al que s'ha de tractar des de un punt de vista del jugador, tant com de programador. No serveix per res tenir el millor codi del món si el resultat per al que hi juga no és satisfactori.

També s'ha arribat a la realització de que el tipus d'arquitectura utilitzat pot ser realment valuós i útil per a jocs de dispositius mòbils que necessiten de poc còmput. D'altra banda s'ha pogut apreciar que aquesta mateixa arquitectura no seria vàlida per jocs a gran escala a cap plataforma. Les limitacions que s'han de mantenir per a utilitzar l'arquitectura com va ser pensada limitaria enormement el rendiment de jocs majors.

AGRAÏMENTS

Primerament m'agradaria agrair al dissenyador Calciumtrice que va dissenyar un pack de imatges per a masmorres, monstres i personatges i les va pujar a la comunitat *OpenGameArt.Org* on es comparteixen dissenys de forma gratuïta per l'ús per part de qualsevol persona, amb qualsevol fi. També m'agradaria agrair al meu company de treball Xevi, una persona apassionada pels jocs, l'arquitectura de software i, en menor mesura, Kotlin per ensenyar-me el llenguatge i com aprofitar el seu potencial.

BIBLIOGRAFIA

[1] Dungeon Crawl. Wikipedia. https://en.wikipedia.org/wiki/Dungeon_crawl
[2] The Bard's Tale launched the Second Wave of RPGs. Jeremy parish. https://www.usgamer.net/articles/the-bards-tale-launched-the-second-wave-of-rpgs
[3] Persona 5 Sells over 2 Million Copies. PlayStation LifeStyle. https://www.playstationlifestyle.net/2017/12/02/persona-5-sales-reach-over-2-million/
[4] GUI Architectures. Martin Fowler. https://www.martinfowler.com/eaaDev/uiArchs.html
[5] Clean Architecture. Robert C. Martin. https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html
[6] Diablo 3 lifetime sales top 30 million units. Polygon. https://www.polygon.com/2015/8/4/9097497/diablo-3-sales-30-million-units
[7] The Binding of Edmund McMillen. GoodTimes. http://goodtimes.sc/cover-stories/the-binding-of-edmund-mcmillen/
[8] Redungeon. Google Play. https://play.google.com/store/apps/details?id=com.nitrome.redungeon&hl=en_US
[9] Pixel Dungeon. Google Play. https://play.google.com/store/apps/details?id=com.watabou.pixeldungeon&hl=en
[10] Kotlin. JetBrains oficial website. https:/kotlinlang.org/
[11] LibGDX. Pàgina oficial. https://libgdx.badlogicgames.com/
[12] OpenGl. Pàgina oficial. https://www.opengl.org/
[13] Realm for Android. Realm Blog. https://realm.io/blog/realm-for-android/
[14] Unreal Engine. Pàgina oficial. https://www.unrealengine.com/

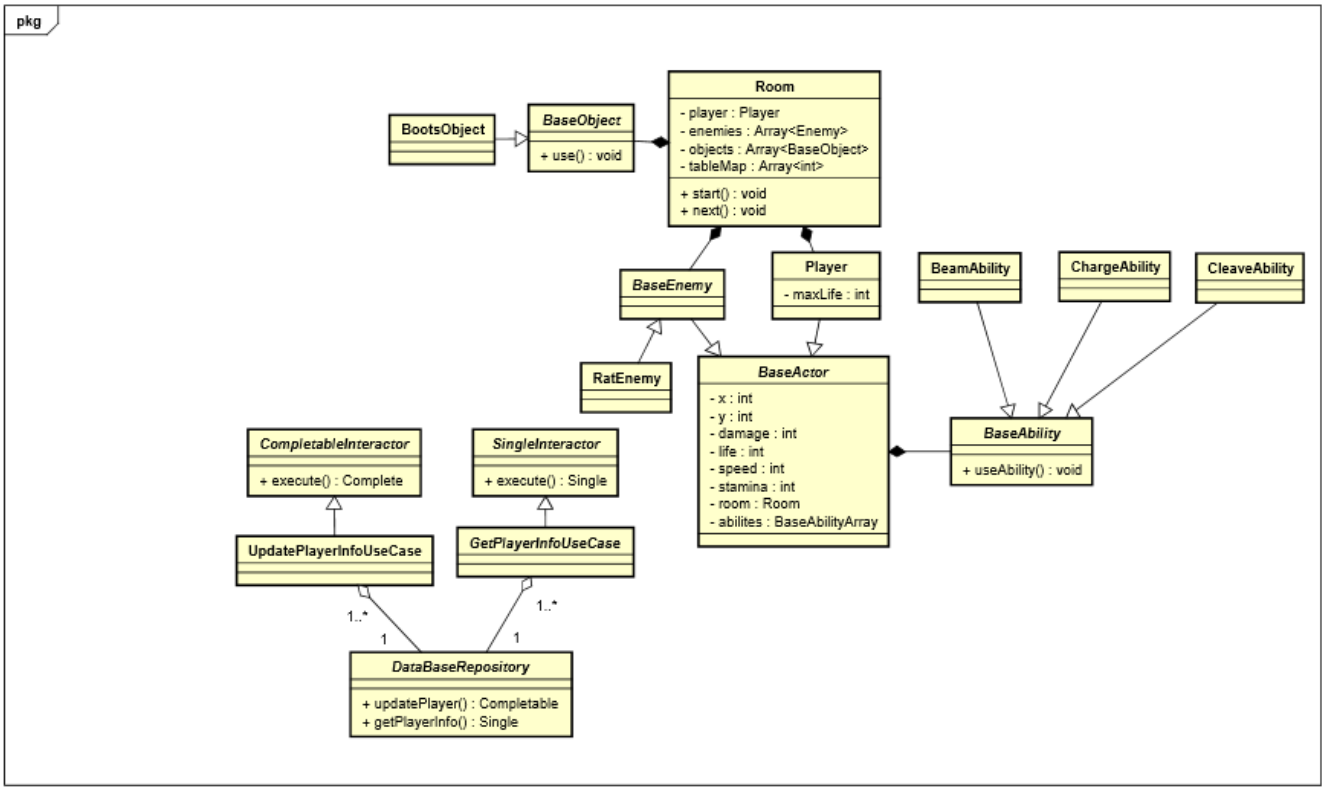
APÈNDIX

A1. GLOSSARI

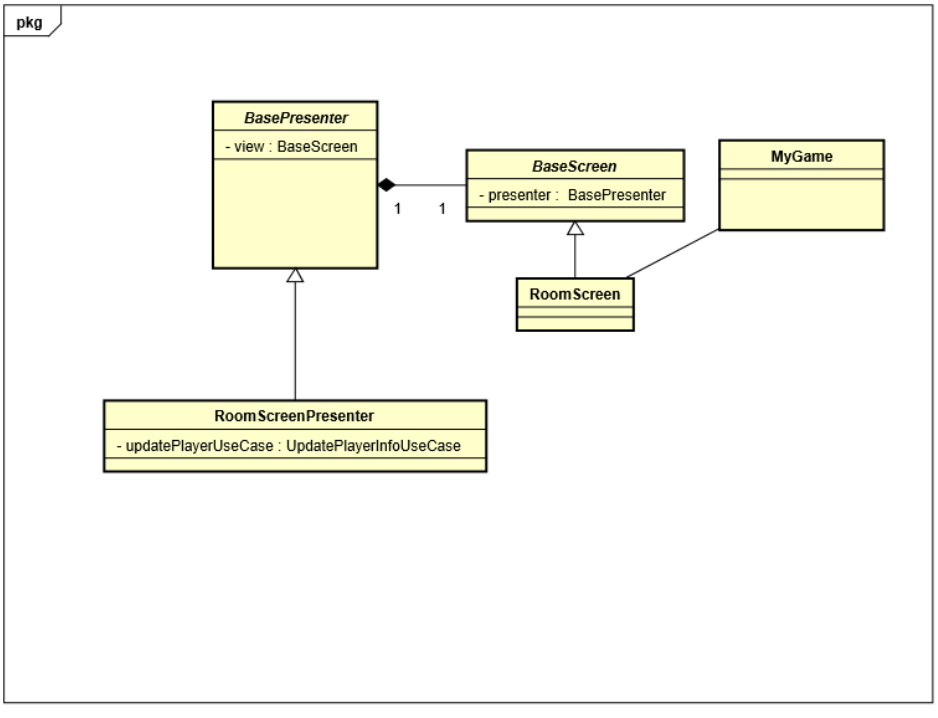
Dungeon Crawler: Gènere de videojocs on el jugador avança a través de un escenari laberíntic recollint tresors i vencent a enemics.
Roguelike: Subgènere dels videojocs Dungeon Crawler caracteritzats per la generació aleatòria de contingut i la pèrdua de progrés quan el jugador perd.
Presenter: Divisió feta per l'arquitectura *model-view-presenter* per a aquella part del codi que gestiona i controla la interacció entre el *model* i la *view*.
View: Divisió feta per l'arquitectura *model-view-presenter* per a aquella part del codi que gestiona que interactua amb el usuari i relleva la lògica al presenter.
Model: Divisió feta per l'arquitectura *model-view-presenter* per a aquella part del codi que executa i guarda la lògica del programa.
Domain: Subdivisió del *model* de l'arquitectura *model-view-presenter* que gestiona i executa la lògica interna definida del programa.
Data: Subdivisió del *model* de l'arquitectura *model-view-presenter* que gestiona les interaccions del programa amb fonts d'informació tal com bases de dades, APIs o arxius externs.

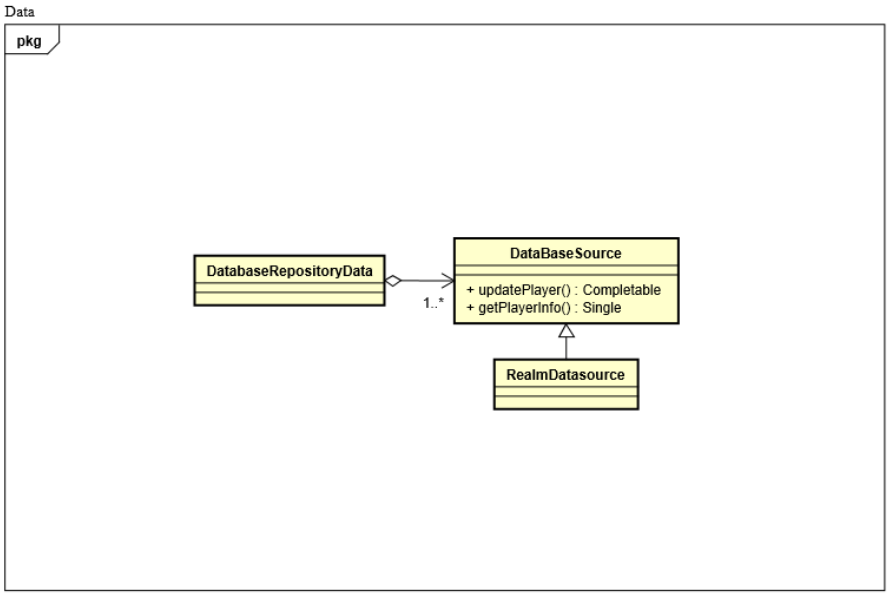
A2. DIAGRAMES DE CLASSES DEL PROJECTE

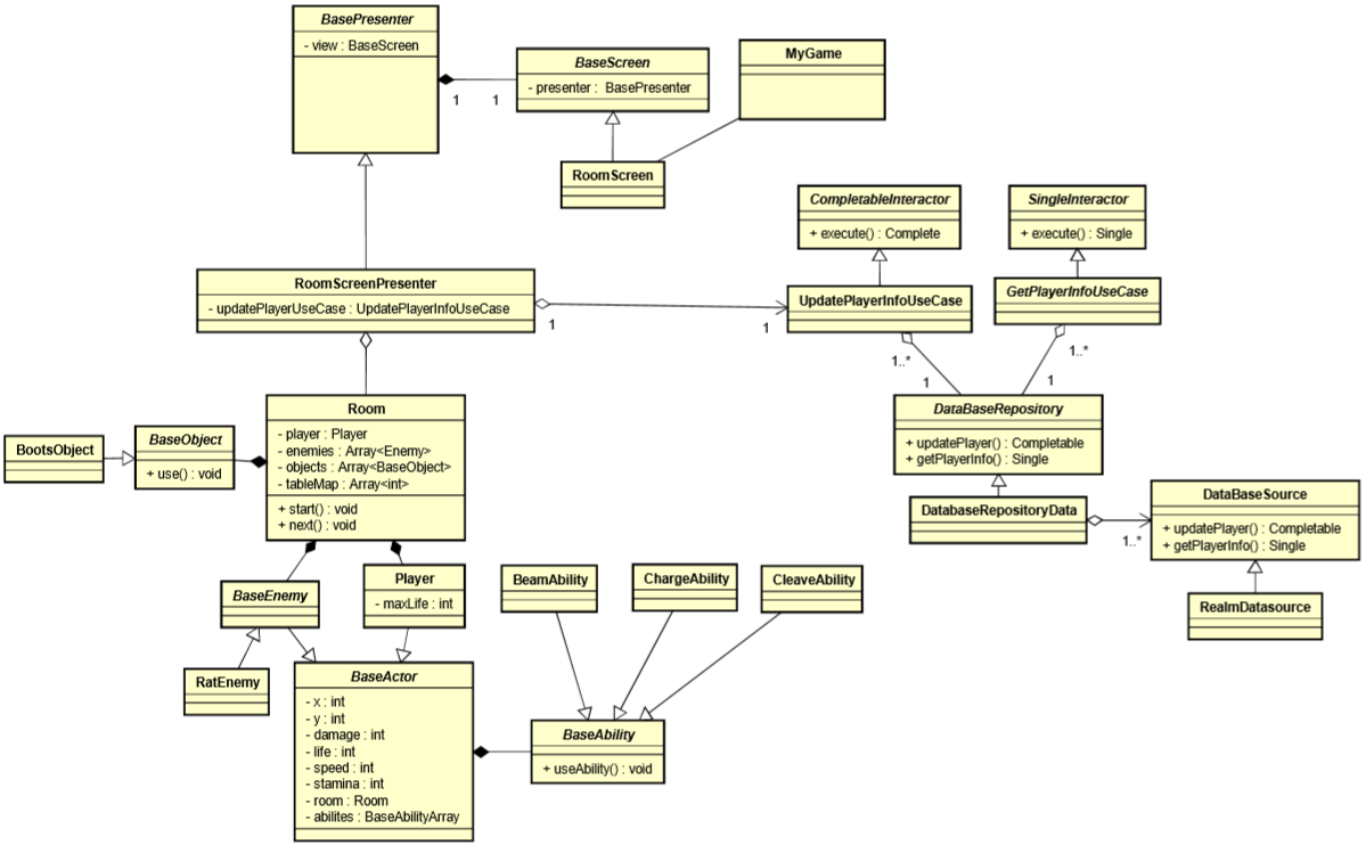
Domain



View + Presenter




Data
pkg
DatabaseRepositoryData
1..*
DataBaseSource
+ updatePlayer() : Completable
+ getPlayerInfo() : Single
RealmDatasource


BasePresenter
- view : BaseScreen
1
1
BaseScreen
- presenter : BasePresenter
MyGame
RoomScreen
CompletableInteractor
+ execute() : Complete
SingleInteractor
+ execute() : Single
RoomScreenPresenter
- updatePlayerUseCase : UpdatePlayerInfoUseCase
1
1
UpdatePlayerInfoUseCase
GetPlayerInfoUseCase
1..*
1
1..*
1
Room
- player : Player
- enemies : Array<Enemy>
- objects : Array<BaseObject>
- tableMap : Array<int>
+ start() : void
+ next() : void
DataBaseRepository
+ updatePlayer() : Completable
+ getPlayerInfo() : Single
BootsObject
BaseObject
+ use() : void
DatabaseRepositoryData
1..*
DataBaseSource
+ updatePlayer() : Completable
+ getPlayerInfo() : Single
RealmDatasource
BaseEnemy
Player
- maxLife : int
BeamAbility
ChargeAbility
CleaveAbility
RatEnemy
BaseActor
- x : int
- y : int
- damage : int
- life : int
- speed : int
- stamina : int
- room : Room
- abilites : BaseAbilityArray
BaseAbility
+ useAbility() : void

A3. FLUX DE LA GENERACIÓ D'HABITACIONS